COMAL-80 Complete Reference (Commodore 64)

This document is intended to be the only COMAL reference an experienced programmer needs to implement, debug and integrate COMAL programs on the Commodore 64. It is condensed but complete: syntax, semantics, packages (graphics, turtles, sprites, sound, I/O), file formats, machine-language integration, memory maps, and clear examples are included.

Use the file as the canonical resource for COMAL on the C-64. Examples assume a COMAL cartridge rev 2.x environment and the standard COMAL packages present on the cartridge.

–

Contents (top-level)

- Introduction & conventions
- Lexical elements and types
- Expressions, operators and precedence
- Statements and structured programming
- Procedures, functions and parameter passing
- Strings and arrays
- Input/Output and file handling
- Packages and the USE/DISCARD model
- Graphics & Turtle (complete API)
- Sprites (complete API)
- Sound & Music (complete API)
- Peripherals: RS-232, parallel port, paddles
- Machine-language modules: LINK, headers, FNDPAR, example
- Memory map and bank-switching
- Error handling and traps
- Useful examples (programs provided in the manual)
- Appendices: ASCII, color codes, error codes (summary)

Conventions used in this manual

- Keywords are shown in UPPERCASE but COMAL is case-insensitive.
- Variable examples: `x`, `counter`, `name$`, `arr(10)`.
- String variables always end with `$` (e.g., `s$`).
- Numeric suffixes: `#` in example source indicates integer variable names used historically (e.g., `i#`) — treat them as identifier characters.

1. Lexical elements

- Identifiers: must start with a letter, then letters/digits/`'` (apostrophe allowed to bind multi-word identifiers, e.g., `wait'key`).
- Numbers: integer (no decimal point), real (decimal and optional exponent). Hex numbers use `$` prefix (e.g., `$A0`), binary with `%` prefix (e.g., `%101010`).
- Strings: delimited by double quotes `"like this"`.
- Comments: `// this is a comment` (the rest of the line is ignored).

2. Types and storage

- COMAL distinguishes: REAL (floating), INTEGER, STRING (text) and ARRAYS (indexed). There is no explicit `DIM` for numeric scalars; arrays and string variables must be dimensioned before use, or COMAL will auto-dimension on first assignment.
- Range: INTEGER is −32768..32767. REAL is machine floating point (~1e-39 to 1e38).

3. Expressions and operators

Priority (highest → lowest): 1. Parentheses `(...)` 2. Exponentiation `EXP(x,y)` or `^` 3. `*`, `/`, `DIV`, `MOD` 4. `+`, `-` (binary); unary `-` applies to single operand 5. Bitwise: `BITAND`, `BITOR`, `BITXOR` 6. Relational: `<`, `<=`, `=`, `<>`, `>=`, `>` 7. Logical: `NOT`, `AND`, `AND THEN`, `OR`, `OR ELSE`

Numeric functions: `INT`, `ABS`, `SGN`, `SQR` (or `SQRT`), `SIN`, `COS`, `TAN`, `ATN`, `LOG`, `EXP`.

String functions: `LEN(s$)`, `LEFT$(s$,n)`, `RIGHT$(s$,n)`, `MID$(s$,start,len)` (COMAL may use `s$(start:len)` in some dialects), `CHR$(n)`, `ORD(ch)`.

Boolean constants: `TRUE` (=1), `FALSE` (=0).

Example:

```
sum := INT(a) + b DIV 2
if (sum > 100) AND (NOT flag) THEN ... ENDIF
```

4. Statements (core)

4.1 Block & flow

- `IF <cond> THEN <stmts> [ELIF <cond> THEN <stmts>] [ELSE <stmts>] ENDIF`
- `CASE <expr> OF WHEN <val> : <stmts> ... OTHERWISE <stmts> ENDCASE`
- `FOR <var> = <start> TO <end> [STEP <s>] DO <stmts> ENDFOR`
- `WHILE <cond> DO <stmts> ENDWHILE`
- `REPEAT <stmts> UNTIL <cond>`
- `LOOP <stmts> [EXIT WHEN <cond>] ENDLOOP`
- `GOTO label:` and `label:` are supported but discouraged (use structured constructs).

4.2 Sequential / data

- `PRINT <items>` optionally `PRINT AT r,c: <items>` for positioned output.
- `PRINT USING <format>` supports formatted numeric output.
- `INPUT "prompt" var` or `INPUT AT r,c: "prompt" var` for interactive data entry.
- `READ varlist`: consumes `DATA` lines; `RESTORE` resets read pointer; `EOD` is the end-of-data condition.

4.3 Program control & system

- `NEW` — clears current program and deactivates packages.
- `RUN` — execute program; `CHAIN "file"` loads and runs another program.
- `SAVE "name"`, `LOAD "name"`, `VERIFY` — persistent file ops.
- `PASS "..."` — pass string to DOS (used for formatting disks etc.).
- `RENUM` / `AUTO` — automatic line-numbering for editing convenience.
- `SCAN` — parse program structure, define procedures; recommended before `RUN`.

4.4 Editing helpers (keyboard)

- Cursor control, `INST/DEL` (insert/delete), `CLR/HOME` clear screen; `CTRL` combos set colors or help editing. The COMAL editor supports line-by-line correction; pressing `RETURN` evaluates the current line.

5. Procedures and functions

5.1 Syntax

- Procedure:

```
PROC name(param1, param2, ...)
  ...
ENDPROC name
```

- Function:

```
FUNC fname(param1, param2)
  ...
  RETURN expr
ENDFUNC fname
```

5.2 Parameters

- Parameters may be passed by value or reference. In module headers (for LINK'ed ML modules) parameters are encoded as `value+type` or `ref+type` with array forms. COMAL checks parameter types at the call boundary.
- Inside ML modules, call `FNDPAR` to get actual storage addresses (see Machine-language section).

5.3 Closed and external procedures

- `CLOSED` procedures are private to the file/module, `EXTERNAL` marks a procedure defined outside current source.

5.4 Example

```
PROC wait'key
  WHILE KEY$ = CHR$(0) DO NULL
ENDPROC wait'key

PROC polygon(n, len)
  FOR i=1 TO n DO
    forward(len)
    right(360/n)
  ENDFOR
ENDPROC polygon
```

6. Strings and arrays

6.1 Strings

- Declare text variables to reserve storage with `DIM name$ OF length` (length in characters). If omitted, COMAL auto-dimensions on assignment.
- Standard operations: concatenation `s3$ = s1$ + s2$`, substring `s$ = s$(start:length)`.

6.2 Arrays

- `DIM arr(dim1, dim2) OF type` reserves an array; COMAL stores necessary metadata (dimensions, bounds) before the data. Indexing is 1-based by default.
- Multi-dimensional arrays are laid out with the last index changing fastest.

Example:

```
DIM names(10) OF 20  // 10 strings each 20 chars
names(1) := "Alice"
```

7. Input/Output and file handling

7.1 File types

- COMAL supports sequential, random and direct file operations; files are used for program storage and data. Disk files and tape files are treated similarly at the COMAL level.

7.2 Common statements

- `OPEN FILE n, "filename", READ|WRITE|APPEND|RANDOM`
- `INPUT FILE n: varlist` — reads item(s) from file `n` into variables.
- `PRINT FILE n: exprlist` — writes items to file `n`.
- `READ varlist` / `DATA` / `RESTORE` — read from in-source `DATA` lines.
- `CLOSE FILE n`
- `CREATE filename` — create new file for writing.

7.3 Example: save & load program (disk)

```
// Save current program to disk
SAVE "polygon"

// Open a sequential data file and write records
OPEN FILE 1, "addresses", WRITE
PRINT FILE 1: name$, street$, city$, phone$
CLOSE FILE 1

// Read back
OPEN FILE 1, "addresses", READ
INPUT FILE 1: name$
CLOSE FILE 1
```

7.4 Low-level DOS interface

- `PASS "n0,mydisk,XX"` — passes the string after `PASS` to the disk operating system (format, catalog, etc.). Example used to format: `PASS "n0,mydisk,XX"`.

8. Packages and modular design

COMAL supports packages: named sets of procedures/functions (and optional initialization) provided by modules. Packages are the standard API to access cartridge-resident or linked functionality (graphics, sound, peripherals).

8.1 Commands

- `LINK "module.obj"` — load an object module (assembler output) into memory (up to 10 modules supported).
- `USE <package>` — define (activate) package procedures; calls package init routine.
- `DISCARD` — remove all non-ROM modules and packages (cannot selectively discard ROM-resident packages).

8.2 Signals

- Modules receive signals on important system events: `POWER` (start-up), `LINK`, `DISCARD`, `NEW`, `RUN`, `CLRTAB`, `ERROR`, `BASIC`, etc. Signal handlers allow modules to register vectors and initialize state.

8.3 Package table and definition (for module authors)

- A module contains a package table listing package names, the address of their procedure name tables, and init routines. Each procedure has a header describing parameter types and the location of machine code.

Detailed binary header formats are required for `LINK`; see Machine-language section.

9. Graphics & Turtle (full)

9.1 Activating graphics

- `USE turtle` or `USE graphics` to enable graphics/turtle orders. `DISCARD` removes packages; `NEW` clears program and packages.
- Modes: `splitscreen` (graphics area + text lines), `fullscreen` graphics via `graphicscreen`/`textecreen` toggles (function keys `F1`,`F3`,`F5` when turtle is active).

9.2 Turtle API (draw with a pen/cursor)

- `forward(L)` / `fd(L)` or `bk(L)` — move turtle leaving a trace (if `pendown`).
- `back(L)` — move backward.
- `left(D)`, `right(D)` — rotate heading.
- `setheading(H)` — heading in degrees (0 up, 90 right by COMAL convention in manual).
- `penup` / `pendown` — do/don't draw.
- `pencolor(C)` — set drawing color code (see color table Appendix).
- `background(C)` — set background color.
- `home` — move turtle to center.
- `clearscreen` / `cs` — clear graphics.
- `hideturtle` / `showturtle` — control cursor visibility.

9.3 Example: regular polygon

```
USE turtle
splitscreen
INPUT "How many sides?": n
INPUT "Length?": len
FOR i:=1 TO n DO
  forward(len)
  right(360/n)
ENDFOR
WHILE KEY$=CHR$(0) DO NULL   // hold graphics
```

9.4 Pixel/bitmap operations

- Graphics package exposes primitives to plot coordinates and set bitmap modes depending on the package level. Use `plot`/`setxy`/`getscreen` functions provided by package (see package manual) or `POKE` directly into bitmap memory at known addresses (advanced).

10. Sprites

Sprites are hardware objects with their own bitmap (24x21 for standard single-color sprites) and attributes (position, color, multi-color mode, enlargement, priority).

Core sprite operations (package `sprites`): - `startsprite(id)` / `stopsprite(id)` — make sprite active/inactive. - `movesprite(id, x, y)` — set sprite position. - `spritecolor(id, color)` — set sprite color. - `saveshape(id, "file")` / `loadshape(id, "file")` — persist shape to disk. - `spritetype(id, mode)` — single/multi color. - `sprsize(id, scale)` — expand sprite (2× enlargement supported). - `collideSprites()` or collision flags — check for collisions between sprites.

10.1 Collision & priority

- Sprite hardware provides collision bits; the package exposes functions to query collisions and sprite priorities which determine which sprite appears on top when overlapping.

11. Sound & Music

Sound is provided via the SID chip using a `sound` package.

11.1 Basic controls

- `soundtype(voice, type)` — choose waveform (pulse, saw, triangle, noise, ringmod).
- `adsr(voice, attack, decay, sustain, release)` — envelope control.
- `note(voice, code$)` — play a note string (e.g., `"C4"`, `"A#3"`).
- `gate(voice, 1|0)` — start/stop the note (open/close gate).
- `volume(n)` — master volume.

11.2 Sequencing & scores

- `setscore(id, tonearray(), adspausearray(), rpausearray())` store sequences; `playscore(id, start, loop)` and `waitscore(id)` manage playback.

11.3 Example: play a note

```
USE sound
soundtype(1, 3)   // choose waveform
adsr(1,6,6,8,6)
note(1, "C4")
gate(1,1)
pause(1)    // function that waits approx 1 second
gate(1,0)
```

12. Peripherals and ports

12.1 RS-232 (serial)

- The COMAL RS-232 interface is TTL-level on the cartridge; a level converter (MAX232 or equivalent) is required for standard ±12V RS-232. Use system routines to configure parity, baud, stop bits (use package docs for exact calls).

12.2 Parallel/User port (DB-25 mapping)

- Port B at a known address is used to read/write byte values. The data direction register (DDR) sets input/output bits. Example addresses (decimal/hex): `portB = 56577 ($DD01)`, `ddr = 56579 ($DD03)` — set DDR to `%00000010` (binary) to make bit1 output and bit0 input.

Example: start/stop train demo (outline)

```
// set DDR: bit0 input, bit1 output
POKE portB'ddr, 2   // decimal 2 => binary 00000010
POKE portB, 2        // set bit1 high to start train
...                 // read bit0 via PEEK(portB) BITAND 1
POKE portB, PEEK(portB) BITAND 253 // clear bit1 to stop
```

12.3 Game ports and paddles

- Control port pins provide digital switches (joystick buttons) and analog POT inputs sampled via timing-capacitor technique. Use `USE paddles` package: `paddle(port, apaddle, bpaddle, abutton, bbutton)` reads paddle and buttons.

Example: basic thermometer reads ADC and converts to temperature via calibration.

13. Machine language integration — LINK, module format, FNDPAR

This section is required reading for anyone integrating assembler modules and writing packages that `LINK` into COMAL.

13.1 Overview

- Steps: assemble module (`.obj`), `LINK "module.obj"`, then `USE package` to define functions/procs.
- Modules can be RAM or EPROM placed; if placed in EPROM the module is marked ROMmed and cannot be discarded.

13.2 Module structure (high level)

- A module contains:
  - header with start address and memory map flags
  - package table: package names + pointer to their procedure name tables + init routine address
  - procedure/function name tables (each name length + string + pointer to procedure header)
  - procedure/function headers describing parameter types and pointer to machine code
  - machine code bodies

13.3 Procedure / function header encoding (concept)

- A procedure header indicates: procedure vs function, function return type (real/int/str), code entry address, number of parameters, and parameter descriptors.
- Parameter descriptors are encoded as: `value+type`, `ref+type`, or `array` variants including dimension counts. Implementers must follow the exact binary format when creating object modules.

13.4 Parameter representation (what ML code receives)

- VALUE+REAL: COPY1 -> mantissa/exponent representation (5 bytes floating format used by COMAL runtime).
- VALUE+INT: COPY1 -> two-byte integer (high/low).
- VALUE+STR: COPY1 -> stored as [maxlen][len][chars…] (use COPY1 pointers accordingly).
- ARRAY parameters: point to descriptor block containing number of indices and element address.

13.5 Calling ML from COMAL — `FNDPAR`

- ML routines must obtain parameter storage addresses at runtime because COMAL manages stack and parameter space dynamically. The system routine `FNDPAR` (Find Parameter) is used as follows:
  - On entry to ML routine, set A = parameter number (1-based), JSR FNDPAR.
  - On return, a pointer (COPY1) contains the parameter address; registers are scrambled and must be preserved as needed.
- Use provided temporary zero-page locations (e.g., `COPY1`, `COPY2`) per COMAL conventions.

13.6 Returning values from ML

- To return integers/reals/strings use the COMAL runtime helper routines (e.g., `pshint` or `pshint`) that convert ML numeric result into COMAL stack representation. For string results, construct the string on the COMAL evaluation stack and return with the stack pointer updated.

13.7 Example: tiny module (conceptual)

- Source (assembly pseudo): module `test` provides `hi()` proc, `add(a#,b#)` (real function), `string$(c$,n#)` (string function). After `LINK "test.obj"` and `USE test`, COMAL can call `hi`, `add`, `string$` directly.

- `FNDPAR` usage: ML `add` obtains parameters addresses, loads integers, performs addition, pushes result back as COMAL REAL or INT using runtime helper call.

14. Memory map & bank switching

14.1 Overview

- The C-64 CPU (6510) addresses up to 64KB. COMAL cartridge and expansion extend functionality by

bank-switching (overlay pages) into the CPU address window.

- COMAL uses cartridge pages mapped into 8000–BFFF (32–48KB region). Cartridge contains 4 pages (16KB each). An optional EPROM expansion can occupy additional pages.

14.2 Useful memory ranges (simplified)

- 0x0000–0x03FF: zero page & system vars
- 0x0400–0x07FF: screen memory (text)
- 0x0800–0x7FFF: program/working RAM (COMAL program, name table, stack)
- 0x8000–0xBFFF: cartridge ROM / overlays (COMAL pages & packages)
- 0xC000–0xFFFF: Kernal/BASIC ROM and character ROM depending on overlays

14.3 Bank switching control

- COMAL provides runtime routines to manipulate overlay registers; modules should use these calls to change maps rather than direct writes unless absolutely necessary.

15. Error handling & TRAP

15.1 TRAP / HANDLER

- `TRAP label` installs a trap handler; `HANDLER ... ENDTRAP` encloses the trap body. Use `REPORT n` from ML or COMAL to raise the named error. `ERR`, `ERRFILE`, `ERRTEXTS` are writable system variables to provide detailed error info.

15.2 Run-time errors and reporting

- Use `REPORT n` to raise predefined COMAL error messages (see Appendix F for mapping). Use `RUNERR` in ML to trigger a COMAL runtime error.

16. Examples (complete and ready to use)

16.1 Draw a square (turtle)

```
USE turtle
splitscreen
FOR i=1 TO 4 DO
  forward(75)
  right(90)
ENDFOR
WHILE KEY$=CHR$(0) DO NULL
```

16.2 Polygon with procedure

```
PAGE
INPUT "How many sides?": number
INPUT "Length of each?": length
USE turtle
splitscreen
polygon(number,length)
wait'key

PROC polygon(number,length)
  FOR i=1 TO number DO
    forward(length)
    right(360/number)
  ENDFOR
ENDPROC polygon

PROC wait'key
  WHILE KEY$=CHR$(0) DO NULL
```

```
ENDPROC wait'key
```

16.3 Train control via parallel port (example extracted from manual)

```
// set up port addresses (example values from manual)
port'b := 56577
port'b'ddr := 56579

PROC set'port'B
  POKE port'b'ddr, 2     // bit0 input, bit1 output
  POKE port'b, 2         // start train (bit1 high)
ENDPROC

PROC stop'train
  POKE port'b, PEEK(port'b) BITAND 253  // clear bit1
ENDPROC

// Use check'light to PEEK(port'b) BITAND 1 to wait for sensor
```

16.4 Simple file record list (basic address list)

See `Address List Demo` example in Appendix H of the original manual — skeleton:

```
// create new file and write records
OPEN FILE 1, "addresses", WRITE
FOR i=1 TO n DO
  PRINT FILE 1: name$(i)
  PRINT FILE 1: street$(i)
  PRINT FILE 1: city$(i)
  PRINT FILE 1: phone$(i)
ENDFOR
CLOSE FILE 1

// read back
OPEN FILE 1, "addresses", READ
INPUT FILE 1: name$
CLOSE FILE 1
```

17. Appendices (quick lookup)

17.1 ASCII & screen characters - See the original COMAL manual Appendix A for the full table — use `ORD()` and `CHR$()` in code to convert.

17.2 Color codes - Use `pencolor(n)` or `background(n)`; consult Appendix B color table. Common codes: `0` black, `1` white, `2` red, `3` cyan, …

17.3 Error codes - COMAL exposes many numeric error messages (0..255). Use `REPORT n` to raise them; trap with `TRAP`.

17.4 Machine-language header summary (for implementors) - Module must provide package table and procedure headers in exact binary format. Function/procedure headers encode parameter kinds and types; arrays include dimension info. See earlier `Machine-language` section.

18. Recommendations & best practices

- `SCAN` often: it validates structure, defines procedures and catches many errors early.
- Break programs into small procedures and `SCAN`/`LIST` regularly.
- Save frequently to both tape and disk for backup while developing.
- Use `USE` selectively to enable only packages you need.

- When writing ML modules, use `FNDPAR` and respected stack/zero-page conventions to avoid corrupting COMAL state.

–

If you want, I will now: - convert this reference into a printable PDF, or - add a one-page cheat sheet that lists the 30 most-used commands and their short forms, or - expand the Machine Language section to include a full, linkable example assembler file compatible with common 6502 assemblers.

Which should I do next?